Computer Physics Communications 134 (2001) 41–46

Computer Physics Communications

www.elsevier.nl/locate/cpc

The HepMC C++ Monte Carlo event record for High Energy Physics ☆

Matt Dobbs [a], Jørgen Beck Hansen [b,*]

[a] *University of Victoria, Department of Physics and Astronomy, P.O. Box 3055, Victoria, Canada V8W 3P6*
[b] *European Laboratory for Particle Physics (CERN), CH-1211 Geneva 23, Switzerland*

Received 22 June 2000

Abstract

HepMC is an Object Oriented event record written in C++ for High Energy Physics Monte Carlo event generators. Many extensions from HEPEVT, the Fortran HEP standard, are supported: the number of entries is unlimited, spin density matrices can be stored with each vertex, flow patterns (such as colour) can be stored and traced, random number generator states can be stored, and an arbitrary number of event weights can be included. Particles and vertices are stored separately in a graph structure, reflecting the evolution of a physics event. The added information supports the modularization of event generators. The event record has been kept as simple as possible with minimal internal/external dependencies. Event information is accessed by means of iterators supplied with HepMC.

PACS: 07.05.Tp; 07.05.Kf

Keywords: Event record; High Energy Physics; C++

1. Introduction

The High Energy Physics (HEP) community is moving towards Object Oriented (OO) computing tools (usually C++): most upcoming experiments are choosing OO software architecture, the development of Pythia 7 [1] in C++ is underway, and the design phase of a C++ version of Herwig [2] has started. Currently no standard OO event record has been adopted by the HEP community. [1] In order for an event record to be accepted it must be simple for the end user to access information, while maintaining the power and flexibility offered by OO design. The HepMC event record has been developed as a proposal to satisfy these criteria and is proposed as a replacement and extension of HEPEVT [5], the Fortran HEP standard.

HepMC is an object oriented event record written in C++ for Monte Carlo Generators in High Energy Physics. It has been developed independent of a particular experiment or event generator. It is intended to serve as both a "container class" for storing events

☆ Available via the following web-address: http://home.cern.ch/mdobbs/HepMC/.

* Corresponding author.
E-mail addresses: Matt.Dobbs@cern.ch (M. Dobbs), Jorgen.Beck.Hansen@cern.ch (J.B. Hansen).

[1] A simple HEPEVT style event record has been proposed by the StdHep [3] group at Fermilab. Several experiments have defined their own event records, see for example [4].

0010-4655/01/$ – see front matter
PII: S0010-4655(00)00189-2

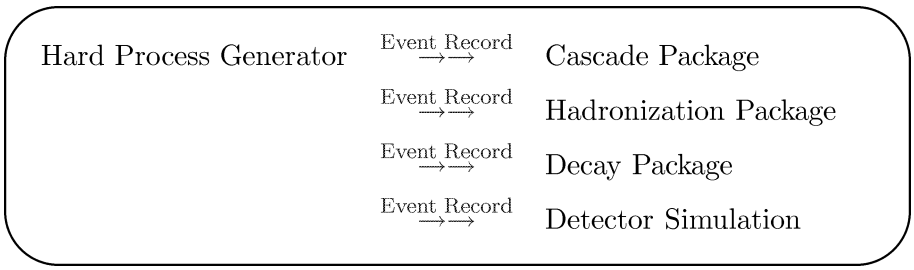


Fig. 1. HepMC supports the concept of modularized event generation (illustrated above) by containing sufficient information within the event record to act as a messenger between two modular steps in the event generation process.

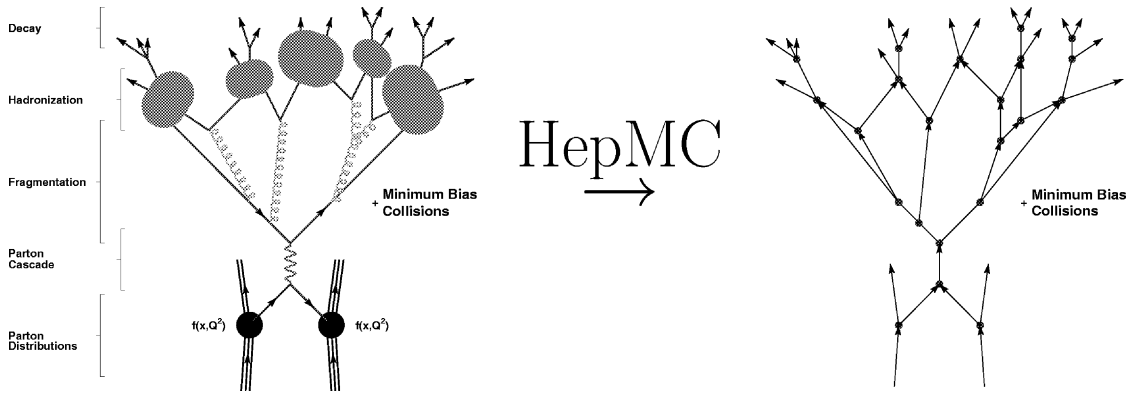


Fig. 2. Events in HepMC are stored in a graph structure (right), similar to a physicist's visualization of a collision event (left).

after generation and also as a "framework" in which events can be built up inside a set of generators. This allows for the modularization of event generators – wherein different event generators could be employed for different steps or components of the event generation process (illustrated in Fig. 1).

2. Structure of the HepMC event record

Physics events are generally visualized using diagrams with a graph structure (Fig. 2, left) which HepMC imitates by separating out particles from vertices and representing them as the edges and nodes of a graph [2] (Fig. 2, right). The extension to multiple collisions is natural – a super-position of many non-connected graphs – and so the event may contain an unlimited number of (possibly interconnected) graphs. Particles and vertices can be combined into any graph structure – there are no restrictions about closed directed cycles, multiple particles connecting vertices, or multiple graph roots.

The components of the event are separated into a set of C++ classes which form the event record and contain the information which is specific to a particular event. A separate modular set of classes forms an index of particle properties, containing the data which are common to all particles of a given type, i.e. charge, mass, lifetime, etc. The HepMC classes are described in this section and the relationships between classes are shown in Appendix A.

2.1. Event

In HepMC, an event is a container of all vertices belonging to the event. Optionally a pointer to the primary root vertex can be stored. To allow the possibility of many processes being generated within the same

[2] Ref. [4] uses a similar structure.

job, a process id can be stored. Extended event features such as a container of event weights and a container for states of random number generators have been implemented such that if left empty or unused performance and memory usage will be similar to that of an event without these features. A container of tags specifying the meaning of the event weights and random number generator state entries is envisioned as part of a higher level class – which describes the complete generation job and is beyond the scope of an event record.

2.2. Vertex

The vertex forms the nodes which link particles into a graph structure. The basic information associated with a vertex is the listing of its incoming and outgoing particles, its position in terms of a Lorentz vector and a possible vertex identifier.

For each vertex a container of weights is included with the intention of storing additional information associated with the vertex, such as amplitude decomposition in terms of colour flow and/or helicity (spin density matrices). It is envisioned that a generator package choosing to assign, e.g., spin density matrices to particle production vertices should provide the functional form of the frame definition for the matrix as a method for interpreting the weights. This implementation is intentionally generic to give maximum freedom to the sub-generators – allowing for different form definitions.

The number of vertices in each event is open-ended.

2.3. Particle

The particle is the basic unit within the event record. A particle is composed of the Lorentz vector, flow, and polarization information. In addition pointers to the particle's production and end vertex are included, which provide an intuitive and fast access to the relationships of a particle. In order to ensure consistency between vertices/particles – these pointers can only be set from the vertex. Thus adding a particle to a vertex as incoming will automatically set the end vertex of the particle to point to that vertex.

Optional extended particle features like flow and polarization have been implemented such that if left empty/unused there is no impact on performance or memory usage.

The flow property of a particle is used to keep track of flow patterns within a graph. The flow pattern information is stored as a series of integer flow codes and indices. Each type of flow (helicity, colour, charge, etc.) is assigned a unique flow index (i.e. colour flow uses index 1 and 2) and each pattern of a given flow type is assigned its own unique integer code. These codes are attached to each particle through which the pattern passes. This is similar to the method used for colour flow in Herwig [2], with the exception that there is no limit to the number of flow types that may be stored. Herwig's method features the possibility of storing non-conserved flow patterns (such as baryon number violation in SUSY models).

Polarization information is stored as the polar and azimuthal angles of a normal three-vector.

The number of particles in each vertex is open-ended.

2.4. Particle data

General particle data information is not a property of an event record, but for completeness a basic particle data table container of particle data has been included in HepMC. There are no dependencies between the particle data objects and the other elements of the event as the access to general particle data is implemented in HepMC as a particle identifier code lookup (the Particle Data Group [6] (PDG) convention is adopted). This keeps the event record modular from the particle data table allowing applications to use the event record independently of the particle data table.

The particle data information available in HepMC is a PDG particle identifier code, name, charge, mass, lifetime ($c \times \tau$) and spin.

3. Accessing information in the event record

3.1. Iterators

Intuitive and fast access to vertices and particles in an event is provided by the use of iterators following the C++ Standard Template Library [7] (STL) style.

Methods to build lists of particles or vertices are *not provided*, as the STL provides these functionalities

with algorithms such as `copy` and iterator adaptors such as `back_inserter` giving a clean generic approach. Using this functionality it is easy to obtain lists of particles/vertices given some criteria – such as a list of all final state particles. Classes which provide the criteria (denoted predicates) are also *not provided*, as the number of possibilities is open ended and specific to the application, however, implementing a predicate is simple.

Two implementations of vertex and particle iterators are provided with HepMC.

The first type of iterators is a simple re-definition of the STL `set::iterator` defined so that all HepMC iterators look similar. These iterators should be used to traverse all vertices or particles in the event exactly once.

The second type of iterators has both a starting point and a range allowing the user to step into a specific part of a particle/vertex graph and obtain targeted information about it. The starting point is the vertex – called the root – from which the iterator was instantiated, and the range is defined relative to this point. The user may choose a range which traverses all parents and/or children (meaning the immediately adjacent particles/vertices going into and/or out of the root) or ancestors and/or descendants (meaning all recursive parents and/or children). The iterator algorithm traverses the graph by converting it to a tree (by "chopping" the edges at the point where a closed cycle connects to an already visited vertex) and returning the vertices in post order. These iterators require more logic than the first type and thus access time is slower (the time required to return one vertex goes like $\log n$, where n is the number of vertices already returned by the iterator).

The particle iterators behaves exactly like the vertex iterators, with the exception that they return particles rather than vertices. As a particle defines an edge or line (rather than a point) in the particle/vertex graph, it is intuitive to define the particle iterator relative to a vertex (point in the graph), thus the starting point (root) is still a vertex, and the range is defined relative to this root.

3.2. Input/output interface

Various input/output strategies are provided with HepMC. The interface for these strategies is specified in an abstract base class. These strategies are capable of input/output of events and particle data tables – as such they depend directly on both the event record classes and particle data table classes.

4. Technical information

HepMC is written within the HepMC:: namespace and consists of 8 core classes (GeneratorEvent, Vertex, Particle, Flow, Polarization, ParticleData, ParticleDataTable, IO_BaseClass) and several utility classes (i.e. IO_HEPEVT, IO_Ascii, IO_PDG_ParticleDataTable, HEPEVT_Wrapper, PythiaWrapper).

The HepMC dependencies have been limited to the C++ Standard Template Library [7] (STL) and the vector classes from the Class Library for High Energy Physics [8] (CLHEP). A simple wrapper for Pythia [9] is supplied to allow the inclusion of event generation examples.

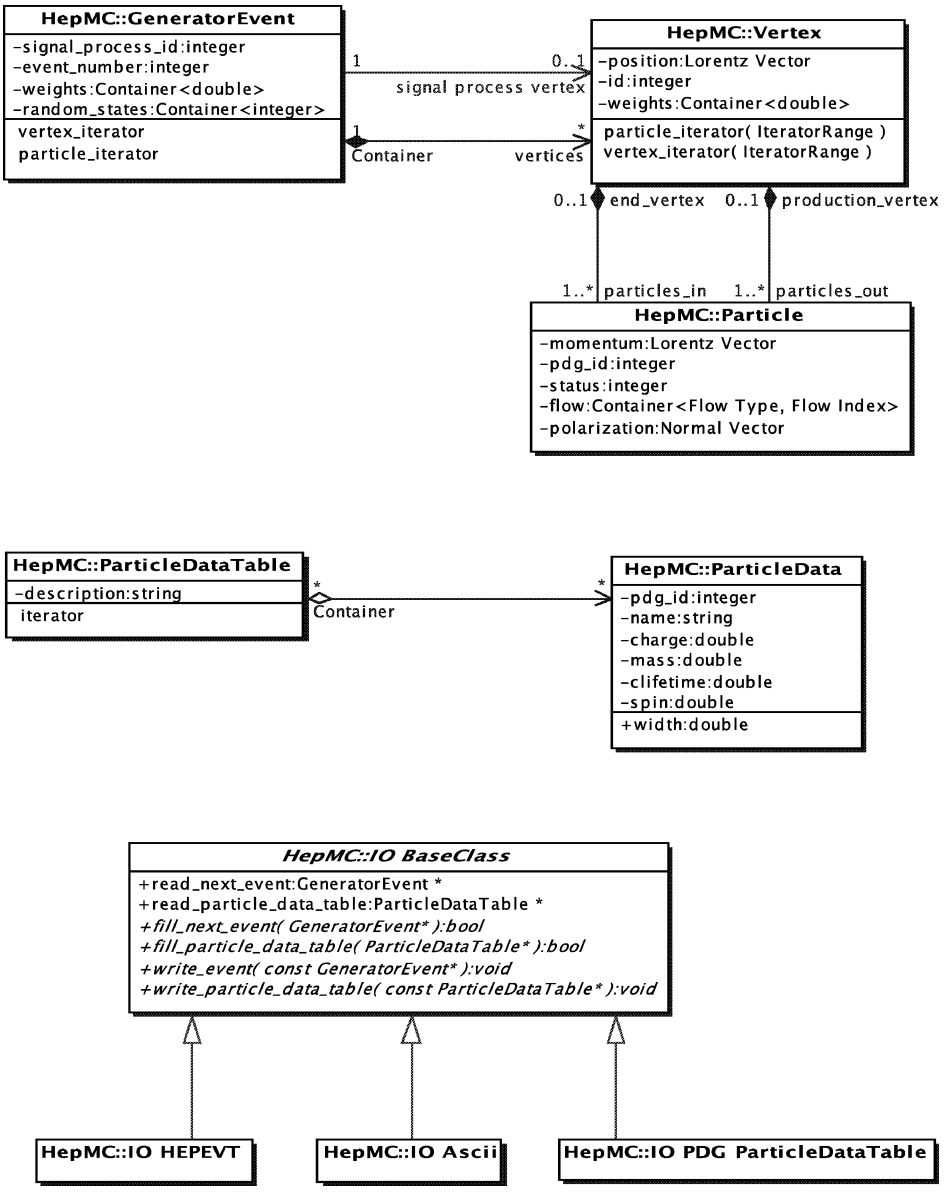
Acknowledgements

The authors would like to thank the ATLAS Collaboration which motivated the HepMC project as a general purpose event record to interface external Monte Carlo event generation packages. In particular we would like to thank Ian Hinchliffe for his support, contributions and discussions. We further extend our thanks to P. Mato, H.T. Phillips, A. Ryd, R.D. Schaffer, M. Smizanska, and B.R. Webber for useful discussions and suggestions.

One of the authors, M. Dobbs, acknowledges support from the Natural Sciences and Engineering Research Council of Canada.

Appendix A

Class diagrams for the event record classes (GeneratorEvent, Vertex, and Particle), the data table classes (ParticleData and ParticleDataTable), and IO classes (IO_BaseClass, IO_HEPEVT, IO_Ascii, IO_PDG _ParticleDataTable) are shown in the Unified Modeling Language [10] notation.



References

[1] L. Lönnblad, Development strategies for PYTHIA version 7, Comput. Phys. Commun. 118 (1999) 213.

[2] G. Marchesini, B.R. Webber, G. Abbiendi, I.G. Knowles, M.H. Seymour, L. Stanco, HERWIG: A Monte Carlo event generator for simulating hadron emission reactions with interfering gluons, Version 5.1, April 1991, Comput. Phys. Commun. 67 (1992) 465.

[3] L. Garren, StdHep 4.08 Monte Carlo Standardization at FNAL, Fermilab PM0091, http://www-pat.fnal.gov/stdhep.html.

[4] S. Protopopescu, MC++ interface, http://www-d0.fnal.gov/newd0/d0atwork/computing/MonteCarlo/MonteCarlo.html.

[5] T. Sjöstrand et al., in: G. Altarelli, R. Kleiss, C. Verzegnassi (Eds.), Z Physics at LEP 1, CERN, Vol. 3 (8), CERN, Geneva, 1989, p. 327.

[6] C. Caso et al., Review of particle physics, European J. Phys. C 3 (1998) 1, http://pdg.lbl.gov/.

[7] A.A. Stepanov, M. Lee, The standard template library, Technical Report HPL-94-34, Hewlett-Packard Laboratories, April 1994, revised July 7, 1995, ftp://butler.hpl.hp.com/stl/.
[8] A Class Library for High Energy Physics (CLHEP), http://wwwinfo.cern.ch/asd/lhc++/clhep/.
[9] T. Sjöstrand, High-energy physics event generation with PYTHIA 5.7 and JETSET 7.4, Comput. Phys. Commun. 82 (1994) 74.
[10] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modelling Language Reference Manual, Addison-Wesley, Reading, MA, 1999; ISBN 0-201-30998-X.